JOURNAL OF COMPLEXITY **18**, 51–86 (2002)
doi:10.1006/jcom.2001.0581, available online at http://www.idealibrary.com on IDEAL®

A Theory of Complexity for Continuous Time Systems

Asa Ben-Hur[1]

Faculty of Industrial Engineering and Management, Technion, Haifa 32000, Israel

Hava T. Siegelmann

Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, Massachusetts 02139
E-mail: hava@mit.edu

and

Shmuel Fishman

Department of Physics, Technion, Haifa 32000, Israel

We present a model of computation with ordinary differential equations (ODEs) which converge to attractors that are interpreted as the output of a computation. We introduce a measure of complexity for exponentially convergent ODEs, enabling an algorithmic analysis of continuous time flows and their comparison with discrete algorithms. We define polynomial and logarithmic continuous time complexity classes and show that an ODE which solves the maximum network flow problem has polynomial time complexity. We also analyze a simple flow that solves the Maximum problem in logarithmic time. We conjecture that a subclass of the continuous P is equivalent to the classical P. © 2001 Elsevier Science (USA)

Key Words: theory of analog computation; dynamical systems.

1. INTRODUCTION

The computation of a digital computer and of its mathematical abstraction, the Turing machine, can be described by a map on a discrete configuration space. In recent years scientists have developed new approaches to computation, some of them based on continuous time analog systems. There is a growing industry of analog VLSI which constructs analog

[1] To whom correspondence and reprint requests should be addressed at current address: Biowulf Technologies, 2030 Addison, Suite 102, Berkeley, CA 94704. E-mail: asa@barnhill technologies.com.

0885-064X/01 $35.00

devices for signal processing and optimization problems. Many of these devices are the implementations of neural networks [1–3] so-called neuromorphic systems [4] which are hardware devices whose structure is directly motivated by the workings of the brain (applications in vision are found for example in [5, 6]). In addition there is an increasing number of theoretical studies of differential equations that solve problems such as linear programming, singular value decomposition, and finding of eigenvectors (see [7] and the references therein). Despite the interest in computation with continuous time systems no theory exists for their algorithmic analysis. The standard theory of computation and computational complexity [8] deals with computation in discrete time and in a discrete configuration space and is inadequate for the description of such systems. This paper describes an attempt to fill this gap.

Our model of a computer is based on ordinary differential equations which converge to attractors. This facilitates a natural definition of the attracting fixed points as the possible outputs of a computation. Convergence to attractors (fixed points, limit cycles, or chaotic attractors) is a property of the class of dissipative dynamical systems which describe most small-scale classical physical systems [9].

In this paper we define a measure of computational complexity which reflects the convergence time of a physical implementation of a given continuous flow, enabling a comparison of the efficiency of continuous time algorithms and discrete time algorithms. In addition, our theory allows the interpretation of various physical and biological phenomena often described by differential equations as processes of computation.

It is important to distinguish our work from the BSS model of computation [10] which is a model of computation *over* the real numbers: the input is real and operations include comparisons between real numbers at unit cost. In our model the parameters of an ODE are the input of the computational machine. In principle, these can be real numbers, but we consider mainly rational inputs in order to have a model which can be compared with the Turing model. Our model *uses* the real numbers since the evolution of an ODE occurs in a continuum.

In their book about the BSS model [11] the authors quote as motivation the words of von Neumann who stated the need for "a detailed, highly mathematical, and more specifically *analytical* theory of automata and of information" [12]. The framework introduced here is an effort in this direction; we believe it strengthens the connection between the theory of computational complexity and the field of continuous dynamical systems. In fact, we propose a subclass of analytically solvable ODEs as a counterpart for the classical complexity class P. This suggests a correspondence between tractability in the realm of dynamical systems and tractability in the Turing model.

Most of the work connecting computation and dynamical systems is aimed at simulating discrete automata. First we note that continuous low-dimensional *maps* in $\mathbb{R}^2$ can be used to simulate the computation of a Turing machine. Simulations with piecewise linear systems were shown in [13, 14] and by analytic functions in [15]. ODEs were used to simulate various discrete time models, thus providing lower bounds on their computational power. Brockett demonstrated how to simulate finite automata by ODEs [16]; Branicky generalized this result to simulate Turing machines [17], thus proving their computational universality. Simulation with piecewise constant functions was shown in [18]. Such constructions retain the discrete nature of the simulated map, in that they follow its computation step by step by a continuous equation. In this paper, on the other hand, we consider continuous systems as they are and interpret their dynamics as a non-symbolic process of computation.

The interest in the theory of computation in continuous time is not new. A survey of some of the previous work appears in [19]. A seminal contribution is the work of Shannon [20] and Pour-El [21] on the so-called general purpose analog computer (GPAC) which was shown to be equivalent in its computational power to a class of differential equations. For a recent extension of this work see [22]. The output of a GPAC computation is determined by the state at a time which is the input of the machine, whereas here output is determined by the asymptotical behavior. The use of continuous time gives the theoretical possibility of "squeezing" even infinite computations into a finite time span [23], enabling the computation of non-recursive functions. Moore's theory of recursive functions over the reals [24] which is an "analog" of classical recursive functions also uses continuous time. Whereas these two approaches are not realizable, we have in mind a physical realization of a differential equation which makes the concept of the computation time take on a well-defined meaning.

The view of the process of computation as a flow to an attractor has been taken by a number of researchers. The Hopfield neural network is a dynamical system which evolves to attractors which are interpreted as memories; the network is also used to solve optimization problems [25]. The continuous time Hopfield network was shown to be equivalent in its computational power to its discrete counterpart in [26]. Brockett introduced a set of ODEs that perform various tasks such as sorting and solving linear programming problems [27]. Numerous other applications can be found in [7]. An analytically solvable ODE for the linear programming problem was proposed by Faybusovich [28]. Our theory is, to some extent, a continuation of their work, in that it provides a framework for the complexity analysis of continuous time algorithms.

Analog computation can be utilized to test possible theoretical limitations of the "physical Church–Turing thesis" [29]. Some theoretical analog

models of computation have the capability of computing beyond the Turing limit [30, 31], but no realizable super-Turing system has been proposed. We do not suggest the current work as a step toward the identification of super-Turing natural systems. Rather, we have the goal of providing an alternative view of computability that has its roots in continuous mathematics and suggest physical systems as readily available special purpose analog computers.

1.1. *Outline of the Paper*

The paper begins with a section providing the required background in dynamical systems. In the following section the issue of defining time complexity for continuous time systems is addressed. Then we introduce the computational model and define a measure of complexity for the subclass of exponentially convergent ODEs. In Section 5 we give the illustrative example of solving the maximum problem in our framework. The more involved example of an algorithm for the maximum network flow problem is given in Section 6. Continuous time complexity classes are defined in Section 7, and a correspondence between the "continuous P" and the classical P is conjectured. In Section 8 possible generalizations of the model are suggested. A short version of this paper has appeared in [32].

2. PRELIMINARIES: DYNAMICAL SYSTEMS

In this section we give the relevant background in dynamical systems. For a detailed introduction the reader is referred to [9, 33, 34]. The flows we consider in this paper are defined by autonomous systems of first order ODEs,

$$\frac{d\mathbf{x}}{dt} = \mathbf{F}(\mathbf{x}), \tag{2.1}$$

where $\mathbf{x}(t)$ is a d-dimensional vector and $\mathbf{F}$ is a d-dimensional vector function of $\mathbf{x}$ with components $(F_1, ..., F_d)$, called a *vector field*. Throughout this paper we use boldface type to denote a vector $\mathbf{a}$, and we denote its components by $a_1, ..., a_d$. We often use the notation $\dot{\mathbf{x}}$ for the time derivative of $\mathbf{x}$. We will assume that $\mathbf{F}$ satisfies a Lipschitz condition to guarantee that a unique solution of (2.1) exists. A *fixed point*, also called an equilibrium point, of (2.1) is a point $\mathbf{x}^*$ such that $\mathbf{F}(\mathbf{x}^*) = 0$. The local stability of $\mathbf{x}^*$ under the flow (2.1) is determined by the linear ODE

$$\frac{d\mathbf{x}}{dt} = M\mathbf{x}, \tag{2.2}$$

where M is the stability operator $D\mathbf{F}|_{\mathbf{x}^*}$, which is the derivative of $\mathbf{F}$ at a point $\mathbf{x}^*$. An eigenvalue λ of M is called *attracting*, *repelling*, or *neutral* according to whether the real part of λ is less than zero, greater than zero, or equal to zero, respectively. A fixed point $\mathbf{x}^*$ is called *attracting*, *unstable*, or a *saddle* if the eigenvalues of M are respectively all attracting, all repelling, or some attracting and others repelling [34]. A fixed point is called *hyperbolic* if all eigenvalues are not neutral. The fixed points of the dynamical systems we consider in this paper will be all hyperbolic. The *basin of attraction* of an attracting fixed point $\mathbf{x}^*$ is the set of points that reach $\mathbf{x}^*$ in the infinite time limit. The boundary of a basin of attraction is defined as usual. For a basin B it is $\bar{B}\backslash B$, where $\bar{B}$ is the closure of B. A hyperbolic attracting fixed point $\mathbf{x}^*$ has the property of *exponential convergence*: for every trajectory $\mathbf{x}(t)$ in the basin of attraction of $\mathbf{x}^*$ there exists strictly positive constants t_0, c, and λ such that for all $t > t_0$, $\|\mathbf{x}(t) - \mathbf{x}^*\| < ce^{-\lambda t}$. This is a result of the solution of Eq. (2.2). In fact, convergence to an attracting fixed point is exponential if and only if it is hyperbolic.

An important class of dissipative systems is the *gradient flows*. A gradient flow is a flow to a local minimum or maximum of a function $E(\mathbf{x})$ and is defined by the equation

$$\dot{\mathbf{x}} = \pm \operatorname{grad} E, \tag{2.3}$$

where the sign of the right-hand side determines whether the flow is to a local minimum or a local maximum (negative sign for minimization). For unconstrained minimization on $\mathbb{R}^d$, grad E is the usual vector of partial derivatives $\partial E/\partial x_i$. For the definition of gradient flows on Riemannian manifolds the reader is referred to [7, 33]. Such a manifold can take into account, for example, the constraint equations of an optimization problem.

3. COMPUTATIONAL COMPLEXITY FOR CONTINUOUS TIME SYSTEMS?

We are interested in ODEs as models of physical systems. For such dynamical systems, the state $\mathbf{x}(t)$ represents the state of the corresponding physical system at time t. The time parameter is then time as measured in the laboratory and has a well defined meaning. Since it is reasonable to associate time with complexity, we suggest using it as a measure of the time complexity of a computation. However, for non-autonomous ODEs (systems with a time dependent vector field $\mathbf{F}(\mathbf{x}, t)$) that are not directly associated with physical systems, the time parameter seems to be arbitrary:[3] if the time variable t of a non-autonomous vector field is

[3] We are grateful to C. Moore and J. Crutchfield for alerting us to this problem.

replaced by another variable s where $t = g(s)$, and $g(s)$ is strictly monotonic, we obtain another non-autonomous system

$$\frac{d\mathbf{x}}{ds} = \mathbf{F}(\mathbf{x}, g(s))\, g'(s). \tag{3.1}$$

The above system will be called the time transformed version of $\mathbf{F}$. If we take, for example, $t = e^s$, then the transformed system computes exponentially faster. This way arbitrary speed-up can be achieved in principle. However, the time transformed system is a *new* system. Only once it is constructed does its time parameter take on the role of physical time and is no longer arbitrary (up to a linear change of the time unit). Therefore, speed-up is a relevant concept only within the bounds of physical realizability. We stress the distinction between linear and non-linear transformations of the time parameter: a linear transformation is merely a change of the units with which time is measured; a nonlinear transformation effectively changes the system itself.

This discussion shows the problem in taking the time parameter of non-autonomous systems as a measure of complexity. We now point out that the class of exponentially convergent autonomous systems is not closed under nonlinear transformations of the time parameter, if we further assume that the vector field is analytic. For autonomous analytic vector fields *convergence to a fixed point is either exponential or polynomial.* When a non-linear speed-up transformation is applied to an exponentially convergent vector field, a non-autonomous system, which converges faster than exponential, results. If one is successful in making the system into an autonomous one, keeping the faster than exponential convergence rate, then it is no longer analytic.

The analyticity property rules out systems such as

$$\frac{dx}{dt} = -(x - x^*)^{1/3}$$

and

$$\frac{dx}{dt} = -\frac{f(x)}{|f(x)|},$$

where $|\cdot|$ is the absolute value. These systems converge in *constant time* to a fixed point due to the fact that these vector fields are not Lipschitz. These are known as *terminal attractors* and were suggested as an efficient model for associative memory [35].

4. THE COMPUTATIONAL MODEL

The model we are about to define is based on a set of autonomous ODEs,

$$\frac{d\mathbf{x}}{dt} = \mathbf{F}(\mathbf{x}), \tag{4.1}$$

where $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{F}$ is a d-dimensional vector field whose dynamics converge exponentially to fixed points. We put special emphasis on gradient flows since their dynamics is simpler to analyze and they provide a convenient problem-solving framework, with many useful examples (see [7]). The motivation for considering flows which converge exponentially to attractors is as follows:

- The existence of attractors provides a convenient and natural way to define the output, as opposed to arbitrary halting regions that arise when simulating a Turing machine by a continuous dynamical system (see [14, 15] for example).
- We consider only exponentially convergent systems since non-exponentially convergent systems do not compute efficiently (see Section 4.7).
- Exponentially convergent systems also have tolerance to noise, or in mathematical terms they are structurally stable; i.e., there exists an ε such that for all perturbations g of norm less than ε, $f+g$ and f are topologically equivalent (there exists a homeomorphism ϕ s.t. $f+g = \phi^{-1} \circ f \circ \phi$). In addition, exponential convergence is the typical convergence scenario in dynamical systems [36].

In the following we describe our interpretation of the dynamics as a process of computation. In this paper the input is taken as the parameters that specify the vector field. It is possible to consider the initial condition as input, as for example is the case in models of associative memory [1] or a Turing machine when viewed as a dynamical system. The latter approach can be problematic and is not taken in this paper; see the discussion in Section 8. In our framework the initial condition is part of the continuous algorithm, and its role is to initiate a trajectory in the basin of attraction of the solution to the problem, and the output of a computation is the attractor of the dynamics.

We introduce our concept of uniformity using the vector field for the maximum problem, which is analyzed in Section 5. This flow has the form $F_i(x_1, ..., x_n) = (c_i - \sum_{j=1}^{n} c_j x_j)\, x_i$, $i = 1, ..., n$. The n parameters c_i are the inputs, and it is seen that a single formula specifies the vector field for all n. Following this example, we formulate the vector field as a fixed *formula*

for inputs of all sizes, and only the length of the various objects in it (vectors, matrices etc.) will vary with the number of inputs.

DEFINITION 4.1. Let $q_1, ..., q_k$ be a fixed finite set of (rational) numerical constants, and let $\Pi = (\pi_1, ..., \pi_n)$ be the parameters of the vector field. Let $\{M_i\}, \{V_i\}$ be finite sets of matrices/vectors respectively with entries in $x_1, ..., x_d; q_1, ..., q_k; \pi_1, ..., \pi_n$, each generated by an algorithm in NC. A *simple algebraic formula* over $x_1, ..., x_d; q_1, ..., q_k; \pi_1, ..., \pi_n; \{M_i\}, \{V_i\}$ is a vector function $\mathbf{F}$ defined by the following operations:

- products ($\prod_i$) and sums ($\sum_i$) of scalars whose number is polynomial in n;
- $+, -, \times$ between vectors, matrices, and scalars; and
- matrix inversion and division between scalars.

The operations we introduced here are all computable in NC [8]. Therefore, since $\mathbf{F}$ is a formula of fixed size, it is also computable in NC. In Section 6 it is shown that with an NC formula as above, one can compute the solution to the P-complete MAXFLOW problem. This suggests that even this simple class of vector fields is sufficient to obtain the complexity class P. This definition can be extended easily to include additional operations and is provided simply as an example of a class of vector fields to which our theory is applicable. In principle, this definition can be extended to include additional operations. In addition, we mention the result of Hoover [37] that computing the fixed point of an NC contracting map[4] is P-complete. This limitation on the vector field ensures that the computational power of the system is based purely on its structure and does not depend for example on constants which serve to increase the computational power, as it is known that neural networks with real weights can compute nonrecursive functions [30].

Given a formula $\mathbf{F}$ one can define the formula $\mathbf{F}' = (\prod_i \pi_i)\,\mathbf{F}$, which is a system that computes exponentially faster than $\mathbf{F}$. We want to assign the same complexity to these two systems since they only differ by a multiplicative factor. This leads us to narrow our discussion to the following class of formulas:

DEFINITION 4.2. A simple algebraic formula $\mathbf{F}$ is said to be *elementary* if $\mathbf{F}$ cannot be written as $G \times \mathbf{H}$ for a scalar formula G which is independent of $x_1, ..., x_d$ and for a vector formula $\mathbf{H}$.

DEFINITION 4.3. A continuous time computer (CTC) is a tuple ($\mathbf{F}$, Π, $\mathbf{x}_0$, Stop) where $\mathbf{F}$ is an elementary formula for a vector field which converges

[4] A contracting map is a special case of a dissipative system: It is a dissipative system that contracts phase space volume in all directions.

exponentially to fixed points. Π are the parameters of the vector field which constitute the input. These are taken to be integer or rational (real inputs can be considered as well, but minor changes in some definitions are required). $\mathbf{x}_0$ is an NC computable initial condition for the flow. "*Stop*" is a procedure for checking that a fixed point being approached is an attractor and that it is computed with enough precision to yield the solution to the problem. As a shorthand we will denote a CTC by the vector field $\mathbf{F}$.

Remark 4.1. We leave the Stop procedure unspecified. This is a component that may vary from system to system. A universally applicable procedure is to halt when a worst case time bound has passed. Halting is discussed in detail in Subsection 4.3.

Remark 4.2. We did not impose restrictions on the relation between the number of variables used to solve a problem instance and its size. This will be done when defining complexity classes. In the example given above for finding the maximum of n numbers, n variables are used.

Remark 4.3. The size (precision) of the initial condition need not be restricted since all initial conditions in a basin of attraction are equivalent in the sense that they produce the same output (possibly with varying computation times), so specifying the initial condition with very high (possibly infinite) precision gives no computational advantage.

4.1. *The Input of a CTC*

The motivation for considering the parameters of the vector field as input is based on optimization problems. In an optimization problem one is given a cost function defined in some state space and is asked to find a state which maximizes (or minimizes) this cost function. The problem of finding a local maximum is solved in our framework with a gradient flow of a cost function E: $\dot{\mathbf{x}} = grad\, E$. The parameters specify an instance of the function E, and the problem is solved by following the trajectory of an initial condition which is in the basin of attraction of the required maximum. One might also take the initial condition as the input, as is the case of the Turing machine or in models of associative memory. However, the computation time for initial conditions near the boundary is unbounded (see an example in Section 5). We comment further on this in Section 8.

To quantify complexity we need a measure of the size of the input. For integer or rational inputs we will consider the *bit-size* measure, denoted by L, which is the number of bits required to encode the input in binary. When real inputs are considered the size of the input is the number of real values.

4.2. *The Output of a CTC*

In our framework we postulate the attractor, or a simple function that is associated with the flow as the output of a computation. In the case of a gradient flow, for example, the output can be taken to be the state which optimizes the cost function or the value of the function itself. In Section 5 we analyze a CTC for finding the maximum of n numbers. This problem can be expressed as a linear programming problem on the simplex: given n numbers represented by a vector $\mathbf{c}$, find the maximum of $\mathbf{c}^T\mathbf{x}$ subject to $\sum_{i=1}^{n} x_i = 1$ and $x_i \geqslant 0$. We show a gradient flow whose fixed points are the vertices of the simplex. These serve as *pointers* to the possible solutions, while the value of the cost function at each vertex represents the possible values of the solution. A gradient flow for the general linear programming problem whose fixed points are exactly the vertices of the solution polytope was introduced in [38] (see Section 6).

The evolution of a CTC reaches an attractor only in the infinite time limit. Therefore, for any finite time we can only compute it to some precision. A computation will be halted when the attractor is computed with enough precision to infer a solution to the associated problem by rounding. When the inputs are integer or rational, the set of fixed points (the possible solutions) will in general be distributed on a grid of some finite precision defined as follows.

DEFINITION 4.4. A CTC $\mathbf{F}$ is said to *require precision* $\varepsilon_p(L)$ if, for every instance Π of size L, every ball of radius $\varepsilon_p(L)$ around the attractor associated with input Π does not contain any other fixed point of $\mathbf{F}$.

Equivalently, a precision $\varepsilon_p(L)$ is required if the attractor associated with every instance of size L can be written as an irreducible rational number/vector with denominator(s) less than $1/\varepsilon_p(L)$.

When the dynamics of a CTC for a problem requiring precision $\varepsilon_p(L)$ reaches the ε_p-vicinity of the attractor, the solution is obtained by rounding the state space variables to the nearest grid point. It is possible to generalize this to allow for the output to be some function of the attractor.

Remark 4.4. For a problem in P with a CTC it is straightforward to show that the required precision is polynomial; i.e., for a problem of size L there exists $k > 0$ such that $\varepsilon_p(L) \leqslant 2^{-L^k}$. In the CTC for the maximum problem the fixed points are the vertices of the simplex which have integer coordinates (see Section 5), and therefore its required precision is $O(1)$.

Remark 4.5. When the inputs of a CTC are real numbers, it may happen that two fixed points are arbitrarily close, and in this case an additional precision parameter is needed. A discussion of real inputs is found in Section 5.5.

4.3. *Halting*

The phase space evolution of a trajectory may be rather complicated, and a major problem is to know when a point approached by the trajectory is indeed the attractor of the dynamics and not a saddle point. To facilitate this decision we define the following.

DEFINITION 4.5. Let $\mathbf{F}$ be a CTC with an attracting fixed point $\mathbf{x}^*$. A *trapping region* of $\mathbf{x}^*$ is a set U containing $\mathbf{x}^*$ such that if $\mathbf{x}(t) \in U$, then for all $t' > t$, $\mathbf{x}(t') \in U$. A trapping region U is said to be *attracting* if there exists a norm $\|\cdot\|$ such that for all t, t', $t' > t$ and $\mathbf{x}(t)$, $\mathbf{x}(t') \in U$ we have that $\|\mathbf{x}(t') - \mathbf{x}^*\| < \|\mathbf{x}(t) - \mathbf{x}^*\|$.

The attracting region is a region in the phase space in which the distance from the attractor is monotonically decreasing relative to some norm.

Remark 4.6. An attracting region as defined above exists for every CTC [33]. When the stability matrix is diagonalizable and has real eigenvalues, e.g., for gradient flows [33], the norm in the definition is the regular Euclidean norm.

We define the convergence time to an attracting region.

DEFINITION 4.6. Let $\mathbf{F}$ be a CTC with an attracting region U; the *convergence time* to the attracting region, $t_c(U)$, is the infimum over all t such that $\mathbf{x}(t) \in U$.

When the computation has reached the attracting region of a fixed point and is also within ε_p of it, namely in $B(\mathbf{x}^*, \varepsilon_p)$, a ball of radius ε_p around $\mathbf{x}^*$, then we can say that the dynamics is near an attracting fixed point and that it is computed with a high enough precision. Thus we define the following.

DEFINITION 4.7. Given a CTC $\mathbf{F}$, let U be an attracting region of an attracting fixed point $\mathbf{x}^*$. Its *halting region*, H, is

$$H = U \cap B(\mathbf{x}^*, \varepsilon_p). \tag{4.2}$$

We can now define the computation time.

DEFINITION 4.8. Let $\mathbf{F}$ be a CTC with a halting region H. Its *computation time* is the convergence time to the halting region, $t_c(H)$.

The convergence time to the halting region is given by

$$t_c(H) = \max(t_c(\varepsilon_p), t_c(U)), \tag{4.3}$$

where $t_c(\varepsilon_p)$ is the convergence time to the ε_p-vicinity of the attractor and $t_c(U)$ is the convergence time to its attracting region, U.

The attracting region of a CTC $\mathbf{F}$ on input Π is some vicinity of the attractor $\mathbf{x}^*(\Pi)$ which is associated with the solution to the problem. Thus computing the attracting region is not easier than solving the problem. Not knowing the attracting region, we do not know when to halt the computation, i.e., when the dynamics has reached $H(\Pi)$. Instead, we specify halting by a bound on the computation time of all instances of size L,

$$T(L) = \max_{|\Pi| = L} t_c(H(\Pi)). \tag{4.4}$$

Thus, on input Π of size L, the computation can be halted after a time $T(L)$. Thus a CTC can be used as the tuple $(\mathbf{F}, \Pi, \mathbf{x}_0, T(L))$, where $T(L)$ takes the role of a halting rule. In practice, one can only obtain a bound on $T(L)$, and efficiency depends on finding tight bounds. Considering systems with an analytical solution allows us to find such bounds. (See the next section for an example.) This halting criterion yields worst case performance in all instances. A halting condition which senses proximity to a fixed point is described in [39].

4.4. *Time Complexity*

As in classical complexity there is the possibility of linear speed-up of a computation. Here linear speed-up is obtained by changing the time unit by a linear transformation $t' = at$ for $0 < a < 1$. This is equivalent to the transformation $\mathbf{F}' = \frac{1}{a}\mathbf{F}$. The computation time in the system $\mathbf{F}'$ is smaller than in $\mathbf{F}$, but only because it is measured in different units. We can make $T(L)$ independent of the chosen time unit by turning it into a dimensionless number. For that we express the time parameter as a multiple of some time unit inherent to the system and choose the *characteristic time scale* that is defined by the rate of convergence to the fixed point.

Let $\mathbf{x}^*(\Pi)$ be the attracting fixed point of $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})$ on input Π. In the vicinity of $\mathbf{x}^*$ the linearization approximation

$$\delta\mathbf{x} = D\mathbf{F}|_{\mathbf{x}^*}\,\delta\mathbf{x}, \tag{4.5}$$

where $\delta\mathbf{x} = \mathbf{x} - \mathbf{x}^*$, holds. Let λ_i be the real part of the ith eigenvalue of $D\mathbf{F}|_{\mathbf{x}^*}$. We define

$$\lambda = \min_i |\lambda_i|. \tag{4.6}$$

λ determines the rate of convergence to the attractor, since in its vicinity

$$|\mathbf{x}(t)-\mathbf{x}^*| \sim e^{-\lambda t}, \tag{4.7}$$

leading to the definition of the characteristic time

$$\tau_{ch}=\frac{1}{\lambda}. \tag{4.8}$$

When the approximation (4.5) holds (the linear regime), in a time $\tau_{ch} \ln 2$ an additional bit of the attractor is computed. Note that $\tau_{ch}=\tau_{ch}(\Pi)$.

A dimensionless complexity measure is given by

$$T'(L)=\frac{T(L)}{\tau_{ch}(\Pi_0)}, \tag{4.9}$$

where Π_0 are the parameters of a fixed instance of the problem, independent of L. T' is invariant under linear transformation of the time parameter.

PROPOSITION 4.1. *Let* $\mathbf{F}$, $\mathbf{F}'$ *be two CTCs related by* $\mathbf{F}'=\frac{1}{a}\mathbf{F}$ *for some constant* $a>0$; *then they have the same time complexity.*

Proof. We denote by primes properties of the system $\mathbf{F}'$. Multiplying the vector field by $\frac{1}{a}$ is equivalent to multiplying the time parameter by a. Therefore the computation times in the two systems are related by $t'_c(\Pi)=at_c(\Pi)$ for every input Π. Let M, M' be the stability operators of $\mathbf{F}$, $\mathbf{F}'$ on input Π, respectively. Clearly, $M'=\frac{1}{a}M$ so that $\tau'_{ch}(\Pi)=a\tau_{ch}(\Pi)$ and in particular for Π_0. We conclude

$$\frac{t_c(H(\Pi))}{\tau_{ch}(\Pi_0)}=\frac{t'_c(H(\Pi))}{\tau'_{ch}(\Pi_0)}.$$

This holds when taking the maximum as well. ∎

Remark 4.7. We now indicate the relation between exponential convergence and efficient computation. We noted that when a system is in the

linear regime, in time $\tau_{ch} \ln 2$ a digit of the attractor is computed. Thus computing L digits requires a time which is $O(\tau_{ch} L)$. We compare this with the case of non-exponential convergence. To get non-exponential convergence at least one eigenvalue of the stability matrix must be zero. Then there is a direction such that $\dot{y} \sim (y - y^*)^\beta$, where $\beta \geqslant 2$. This yields polynomial convergence, $|\mathbf{x}(t) - \mathbf{x}^*| \sim t^{-\beta+1}$. In order to compute $\mathbf{x}^*$ with L significant digits, we need to have $|\mathbf{x}(t) - \mathbf{x}^*| < 2^{-L}$, or $t > 2^{L/(\beta-1)}$ for an exponential time complexity. Note that requiring the vector field to be analytic rules out convergence which is faster than exponential.

5. A CTC FOR THE MAXIMUM PROBLEM

We demonstrate our approach with a simple CTC for the MAX problem, which is the problem of finding the maximum of n numbers.

5.1. *Formulation as an Optimization Problem*

Let the numbers be $c_1, ..., c_n$, and define the linear cost function

$$f(\mathbf{x}) = \mathbf{c}^T \mathbf{x}. \tag{5.1}$$

The MAX problem can be phrased as a constrained optimization problem: Find the maximum of f subject to the constraints

$$\sum_{i=1}^{n} x_i = 1, \qquad x_i \geqslant 0. \tag{5.2}$$

This is recognized as the linear programming problem on the $n-1$ dimensional simplex

$$\Delta_{n-1} = \left\{ \mathbf{x} \in \mathbb{R}^n : x_i \geqslant 0, \sum_{i=1}^{n} x_i = 1 \right\}. \tag{5.3}$$

We use the vector field

$$F_i = \left(c_i - \sum_{j=1}^{n} x_j c_j \right) x_i, \tag{5.4}$$

which is the gradient of the function f on Δ_{n-1} relative to a Riemannian metric which enforces the positivity constraints for points in Δ_{n-1} [7]. This flow is a special case of the Faybusovich vector field [38], which is analyzed in Section 6.

5.2. *Analysis of the Flow*

We denote by $\mathbf{e}_1, ..., \mathbf{e}_n$ the standard basis of $\mathbb{R}^n$. It is easy to verify that the only fixed points of $\mathbf{F}$ are the n vertices of the simplex $\mathbf{e}_1, ..., \mathbf{e}_n$. We begin with the case of a unique maximum. For the purpose of analyzing the flow suppose that $c_1 > c_2$ and $c_2 \geqslant c_j$, $j = 3, ..., n$. Under this assumption the flow converges exponentially to $\mathbf{e}_1$, as witnessed by the solution to the equations $\dot{\mathbf{x}} = \mathbf{F}$,

$$x_i(t) = \frac{e^{c_i t} x_i(0)}{\sum_{j=1}^{n} e^{c_j t} x_j(0)}, \tag{5.5}$$

where $x_i(0)$ are the components of the initial condition. It is important to note that the analytical solution does not help in determining which of the fixed points is the attractor of the system: one needs the solution to the specific instance of the problem for that. Thus the analytical solution is only a formal one, and one has to follow the dynamics with the vector field (5.4) to find the maximum.

To avoid the problem of flow near saddle points exemplified in Fig. 1 we choose an initial condition at the center of the simplex,

$$\frac{\mathbf{e}}{n} = \frac{1}{n}(1, ..., 1)^T. \tag{5.6}$$

This subsection can be summarized by the following interim result.

LEMMA 5.1. *$(\mathbf{F}, \mathbf{c}, \mathbf{e}/n, T(L))$ is a CTC for MAX, where $T(L)$ is a bound on the convergence time to the halting region.*

In the following subsection we conclude the analysis of the algorithm with a derivation of a bound $T(L)$.

5.3. *The Computation Time*

Because of the constraint $\sum_{i=1}^{n} x_i = 1$, the flow has $n-1$ independent variables, which we choose as $x_2, ..., x_n$. The asymptotic behavior of the

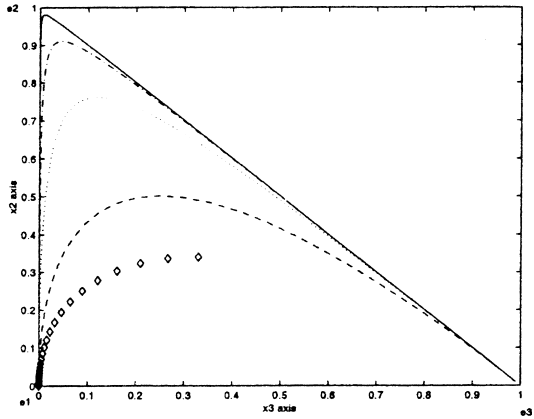

FIG. 1. Phase space behavior of the flow generated by (5.4) on the two-dimensional simplex with $c_1 > c_2 > c_3$. A projection onto the plane of the x_2 and x_3 coordinates is shown. A number of trajectories which start near the minimum vertex $\mathbf{e}_3$ are plotted. Such trajectories reach the vicinity of the saddle point $\mathbf{e}_2$ before flowing in the direction of the maximum which is projected onto the origin. The trajectory of the interior initial condition $\mathbf{e}/3 = \frac{1}{3}(1, 1, 1)$ is denoted by diamonds.

solution for the independent variables is $x_i(t) \sim e^{(c_i - c_1)t}$, $i = 2, ..., n$. Therefore the time scale of $\mathbf{F}$ is

$$\tau_{ch} = \frac{1}{c_1 - c_2}. \tag{5.7}$$

This can also be obtained from a linearization of $\mathbf{F}$ in the vicinity of $\mathbf{e}_1$.

LEMMA 5.2. *The problem precision for MAX instances with a unique solution satisfies* $\varepsilon_p = 1/2$ *and*

$$t_c(\varepsilon_p) \leqslant \tau_{ch} \log n.$$

Proof. The possible solutions (vertices of the simples) are integer and therefore $\varepsilon_p = 1/2$. From the constraint $\sum x_i = 1$ we have that $\|\mathbf{x} - \mathbf{e}_1\| < 1/2$ if $x_j \leqslant 1/2n$ for $j > 1$. The time to reach the ε_p-vicinity of $\mathbf{e}_1$ is thus obtained by solving for t in the equation $x_j(t) \leqslant 1/2n$. ∎

PROPOSITION 5.1. *Let* $\mathbf{c} \in \mathbb{N}^n$ *be an instance of MAX with a unique maximum, then*

$$t_c(H(\mathbf{c})) \leqslant \tau_{ch}(\log \tau_{ch} + \log n + \log c_2). \tag{5.8}$$

Proof. We compute a bound on the convergence time to the attracting region. The attracting region of the problem is the set in which $\dot{x}_i < 0$ for $i > 1$. By the constraint $\sum_{i=1}^{n} x_i = 1$, x_1 is then guaranteed to increase. The convergence time to the attracting region is a time t such that for all $t' > t$, $\dot{x}_i(t') < 0$ for $i > 1$. From the flow equations and the non-negativity of the x_i's we have that $\dot{x}_i < 0$ for $i > 1$ iff

$$\sum_{j=1}^{n} c_j x_j > c_i, \qquad i = 2, ..., n. \tag{5.9}$$

Since by assumption $c_2 \geqslant c_j$ for $j > 2$, this is equivalent to

$$\sum_{j=1}^{n} c_j x_j > c_2. \tag{5.10}$$

Inserting the analytical solution (5.5), specialized to the initial condition $\mathbf{e}/n$ (see (5.6)), we obtain

$$\sum_{j=1}^{n} c_j e^{c_j t} > c_2 \sum_{j=1}^{n} e^{c_j t}.$$

This can be obtained also by differentiating the solution with the demand that $\dot{x}_i < 0$ for $i > 1$. We rewrite the equation as

$$(c_1 - c_2)\, e^{c_1 t} > \sum_{j>1} (c_2 - c_j)\, e^{c_j t}.$$

This inequality holds for t_c which satisfies

$$(c_1 - c_2)\, e^{c_1 t_c} > (n-2)\, c_2 e^{c_2 t_c},$$

from which we obtain

$$t_c(U) < \frac{1}{c_1 - c_2} \log \frac{(n-2)\, c_2}{c_1 - c_2} \leqslant \tau_{ch}(\log \tau_{ch} + \log n + \log c_2), \tag{5.11}$$

when $(n-2)\, c_2 > c_1 - c_2$. If this is not the case, the inequalities (5.9) hold already for the initial condition; i.e., the initial condition itself is in the attracting region, and $\dot{x}_i(t) < 0$ for all $t \geqslant 0$ and $i > 1$.

The maximum of (5.11) and (5.2) gives the required bound on $t_c(H(\mathbf{c}))$. ∎

Remark 5.1. The condition which defines the attracting region equation (5.10) can be expressed as $f(\mathbf{x}) > f(\mathbf{e}_2)$. When this is satisfied it is satisfied for all subsequent times, since the cost function of a gradient flow is increasing along a trajectory. Or, in other words, the set $\{\mathbf{x} \in \Delta_{n-1} : f(\mathbf{x}) > f(\mathbf{e}_2)\}$ is a trapping set (positively invariant set in the language of dynamical systems) of the flow. This condition is appropriate for any gradient flow. Also, the condition on proximity to the attractor can be substituted here for proximity of the cost function to its value on the attractor.

5.3.1. *Non-unique Maximizer.* In the case when the maximum c_i is not unique any combination of the maximal vertices is a maximum of f. Suppose that the maximum has multiplicity k and that $c_1 = c_2 = \cdots = c_k$, $c_k > c_j$, $j = k+1, ..., n$; then on the initial condition $\frac{1}{n}(1, ..., 1)^T$ the algorithm converges to the point $\frac{1}{k}(1, ..., 1, 0, ..., 0)^T$. Moreover, the dynamics is restricted to a lower dimensional manifold with the additional constraints $x_1 = x_2 = \cdots = x_k$. By a derivation similar to that for (5.11) one obtains in this case

$$t_c(U) < \frac{1}{c_1 - c_{k+1}} \log \frac{(n-k-2)\, c_{k+1}}{k(c_1 - c_{k+1})}. \tag{5.12}$$

Thus the bound (5.11) holds with c_2 replaced with c_{k+1}. In the case of a non-unique maximizer the required precision is changed from $O(1)$ to $O(1/n)$, but all this does is to double the bound on $t_c(\varepsilon_p)$. Therefore the bound on the computation time, (5.8), is essentially unchanged.

Remark 5.2. In the extreme case when all the c_i's are equal $\mathbf{F} = 0$. This does not present a problem, since then each c_i is maximal. So when one encounters a system with no dynamics, the computation can be halted immediately.

To combine the expressions (5.11) and (5.12) we introduce the notation

$$c_{(1)} = \max_i c_i$$

$$c_{(2)} = \begin{cases} \max_i \{c_i : c_i < c_{(1)}\} & \text{not all } c_i \text{ are equal} \\ c_{(1)} & \text{otherwise} \end{cases}$$

so that the expression (5.7) for τ_{ch} is replaced with

$$\tau_{ch} = \frac{1}{c_{(1)} - c_{(2)}} \tag{5.13}$$

and

$$t_c(H) < \tau_{ch}(\log \tau_{ch} + \log 4n + \log c_{(2)}). \tag{5.14}$$

5.4. *Complexity of the Algorithm*

We now proceed to analyze the problem size dependence of the above bound on $t_c(H)$, Eq. (5.14). This dependence varies according to the set from which the input is taken through the problem size dependence of the characteristic time scale. We recall that the bit-size complexity of a problem instance, L, is the length of the binary encoding of the input. This is appropriate for models over the integers or rationals. When the inputs are integers, $L = \sum_{i=1}^{n} (1 + \log(c_i + 1))$. As in the BSS model of computation over the real numbers [11], for real inputs the size of a problem is the number of inputs. The complexity for different input sets is summarized in Table I and is easily derived from expressions (5.13) and (5.14).

We now discuss the results of Table I. The second row of the table considers the case of inputs which are integer or rationals with bounded denominators (rational numbers which can be specified with fixed precision). In these two cases we obtain sub-linear (logarithmic) complexity, which may seem surprising. However, note that the variables of a CTC can be considered as processing units. Therefore when the number of variables is an increasing function of the size of the input, as in the CTC we presented for MAX, the model becomes inherently parallel. There are various models of parallel computation, and the complexity of MAX is different in each of them: in the PRAM model its complexity is $O(\log \log n)$ [40]; in the circuit model its complexity is constant or logarithmic depending on the fan-in (unbounded or constant, respectively). It is not clear which model ours is most closely related to.

TABLE I

Bounds on τ_{ch} and T as a Function of the Input Set.

Input space	Problem size	Bound on τ_{ch}	T
Bounded integers or bounded rationals with bounded denominators	$L = O(n)$	$O(1)$	$\log n$
Integers or rationals with bounded denominators	L	$O(1)$	$\log L$
Rationals	L	L	$L \log L$
Reals	n	unbounded	poly in condition number logarithmic in n

For general rational inputs (third row of the table) the complexity is again comparable to that obtained by a naive algorithm in the Turing model: for a problem of size L, the comparison of two numbers requires $O(L)$ bit operations. This is multiplied by the number of comparisons required between numbers.

For real inputs there is no bound on τ_{ch} and hence on the computation time. In such cases we consider complexity as a function of a "condition number" defined in the next subsection. Note that in the BSS model the MAX problem with real inputs can be solved efficiently because comparisons between real numbers are allowed.

5.5. *Complexity and the Distance from Ill-Posed Problems*

In the numerical analysis literature it is common to consider the condition number of a problem instance as a measure of its difficulty [41]. The condition number is defined as the norm of the condition operator which measures the effects of infinitesimal perturbations in the input on the result of a computation [42]. In many settings the condition number is shown to be the reciprocal of the distance to the set of ill-posed problems, where a problem instance is said to be ill-posed if its solution is not unique. Such results are called "condition number theorems." In combinatorial optimization the output does not depend continuously on the input so the condition operator cannot be defined in this manner. In the context of the linear programming problem, Renegar [43] postulates the condition number as the inverse of the distance to the set of ill-posed problems. Such a definition is motivated by these condition number theorems.

As in combinatorial optimization, the output of a computation of a CTC is not necessarily a continuous function of the input. Therefore a condition number cannot be defined in terms of a condition operator as above. Instead we define τ_{ch} as the condition number, which provides a direct relation between conditioning and complexity. For vector fields which have a stability matrix with real eigenvalues (e.g., gradient flows) τ_{ch} can be expressed as

$$\tau_{ch} = \|(D\mathbf{F}|_{\mathbf{x}^*})^{-1}\|, \tag{5.15}$$

where $\|\cdot\|$ is the operator norm: $\|A\| = \max_{\mathbf{x}\neq 0}(A\mathbf{x}/\|\mathbf{x}\|)$. This expression shows the similarity in the definition of τ_{ch} and a condition number, where the inverse of the stability operator plays the role of condition operator. In the following we argue the plausibility of a condition number theorem type of result for τ_{ch}.

For simplicity we assume that the fixed point structure of the system is fixed for a fixed number of parameters, so that the role of the input is to

determine which of the fixed points is the attractor of the system, as was the case in the CTC for MAX. Let Π be the input of an ill-posed instance. We further suppose that the stability matrix depends continuously on the input. Therefore in any neighborhood $\mathcal{U}$ of Π there exist $\mathbf{x}_1 \neq \mathbf{x}_2$ which are the attractors on inputs in $\mathcal{U}$. Let $\Pi_1, \Pi_2 \in \mathcal{U}$ be inputs with the associated attractors $\mathbf{x}_1, \mathbf{x}_2$ respectively. Suppose that $\mathbf{x}_1$ and $\mathbf{x}_2$ are connected by an invariant manifold $\mathcal{W}$, which on input Π_1 is part of the stable manifold of $\mathbf{x}_1$ and on input Π_2 is part of the unstable manifold of $\mathbf{x}_1$. It is reasonable to suppose that on input Π the attractor of the system is either $\mathbf{x}_1$ or $\mathbf{x}_2$ or that it is some point in $\mathcal{W}$ which depends on the initial condition. We show that for such points the stability matrix is singular on input Π.

First we consider the case that the attractor on input Π is either $\mathbf{x}_1$ or $\mathbf{x}_2$. The eigenvalues of the stability matrix at $\mathbf{x}_1$ on input Π_1 are all negative, and on input Π_2 the stability matrix at $\mathbf{x}_1$ has at least one positive eigenvalue. By the continuity of the stability matrix as a function of the input, we obtain that on input Π the stability matrix at $\mathbf{x}_1$ has an eigenvalue zero. The same argument holds if the attractor is at $\mathbf{x}_2$.

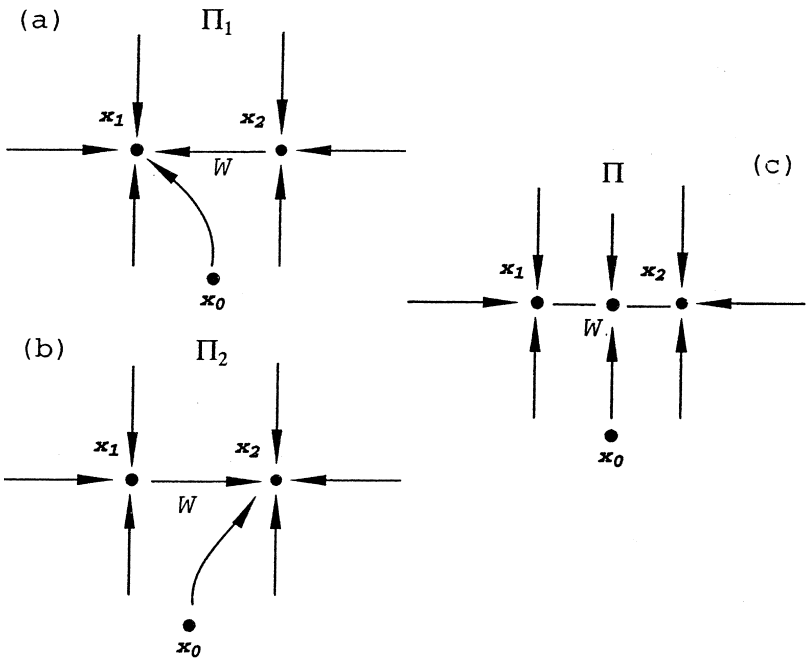

FIG. 2. In the vicinity of an ill-posed instance Π the system has two different attractors: (a) $\mathbf{x}_1$ on input Π_1 and (b) $\mathbf{x}_2$ on input Π_2. (c) On input Π the attractor is either $\mathbf{x}_1$, $\mathbf{x}_2$, or a point in $\mathcal{W}$.

Now consider the case that the attractor on input Π and initial condition $\mathbf{x}_0$ is a point $\mathbf{x}^3$ in $\mathscr{W}$. On input Π_1 the eigenvalues of the stability matrix at a $\mathbf{x}_3$ in directions tangent to $\mathscr{W}$ are negative, whereas on input Π_2 these eigenvalues are positive. Thus on input Π the eigenvalues of the stability matrix at $\mathbf{x}_3$ in directions tangent to $\mathscr{W}$ are zero.

We thus obtain an inverse relation between the distance to an ill-posed instance and its condition number: when an instance is close to an ill-posed instance, the stability matrix is close to singular, with a large τ_{ch}.

In the case of the CTC for MAX a condition number theorem, stating that τ_{ch} is the reciprocal of the distance from the set of ill-posed problems, can be shown. The-set of ill-posed problems of MAX is the set of vectors whose maximum is not unique,

$$\Sigma_n = \{\mathbf{c} \in \mathscr{I}_n : \exists i, j, i \neq j \text{ such that } c_i = c_j = c_{(1)}\}, \tag{5.16}$$

where $\mathscr{I}_n$ is the set from which the inputs are taken. A problem is "close" to ill-posed if $c_{(1)} - c_{(2)}$ is small, leading to long computation times, since this is the inverse of the characteristic time scale. The distance of an instance $\mathbf{c}$ of MAX to the set of ill posed problems is

$$d(\mathbf{c}, \Sigma_n) = \min_{\mathbf{c}' \in \Sigma_n} \|\mathbf{c} - \mathbf{c}'\|. \tag{5.17}$$

It is easy to verify the following.

Claim 5.1. Let $\mathbf{c}$ be an instance of MAX, then

$$d(\mathbf{c}, \Sigma_n) = \frac{1}{\|(D\mathbf{F}|_{\mathbf{e}_{(1)}})^{-1}\|} = \frac{1}{\tau_{ch}} = c_{(1)} - c_{(2)}, \tag{5.18}$$

where $\mathbf{e}_{(1)}$ is the vertex corresponding to $c_{(1)}$.

The discussion in this subsection sheds light on the complexity results in Table I. When the inputs are integers, the distance of a problem instance from the ill-posed set is large, giving highly efficient computation, while for general rational inputs this distance can be exponentially small. For real inputs the distance can be arbitrarily small, with unbounded computation time.

Remark 5.3. In Subsection 5.3.1 we have shown that the CTC for MAX has the special property that it performs well even for ill-posed instances. This may seem surprising in view of the above discussion, but it arises because of the symmetry properties of the vector field: When the stability operator becomes degenerate, the system converges exponentially to a

convex combination of the vertices which correspond to $c_{(1)}$. An ill-posed instance with multiplicity k of the maximum can be viewed as a non-degenerate instance in $\mathscr{D}_{n-(k-1)}$, and its ill-posedness should be measured in $\Sigma_{n-(k-1)}$. And, indeed, the bound (5.14) shows that the characteristic time scale of such a problem is $c_{(1)}-c_{(2)}$, which is the distance of the reduced problem from the set $\Sigma_{n-(k-1)}$. Such behavior is also observed in the algorithm introduced in the next section.

6. A CTC FOR THE MAXFLOW PROBLEM

In this section we present a CTC for the maximum network flow problem (MAXFLOW) which is based on a flow for linear programming. The flow does not yield an algorithm which is efficient in a worst case analysis for general linear programming instances. However, it turns out to be efficient for MAXFLOW and other related problems which can be expressed as linear programming problems. We begin with the definition of the linear programming problem (LP) and introduce the Faybusovich vector field [38] for solving it.

The *standard form* of the linear programming problem is

$$\max\{\mathbf{c}^T\mathbf{x} : \mathbf{x}\in\mathbb{R}^n,\ A\mathbf{x}=\mathbf{b},\ \mathbf{x}\geqslant\mathbf{0}\}, \tag{6.1}$$

where $\mathbf{c}\in\mathbb{R}^n$, $\mathbf{b}\in\mathbb{R}^m$, $A\in\mathbb{R}^{m\times n}$, and $m\leqslant n$. Assuming that a bounded optimal solution exists, the constraint set generated by the constraints in (6.1) is a polyheder, and the maximum is obtained at one of its vertices. Let $B\subset\{1,\ldots,n\}$, $|B|=m$, and $N=\{1,\ldots,n\}\backslash B$, and denote by $\mathbf{x}_B$ the coordinates with indices from B and A_B, the $m\times m$ matrix whose columns are the columns of A with indices from B. A vertex of the LP problem is defined by a set of indices B which is called a *basic set* if

$$\mathbf{x}_B=A_B^{-1}\mathbf{b}\geqslant\mathbf{0}. \tag{6.2}$$

Its components are $\mathbf{x}_B$ and $\mathbf{x}_N=\mathbf{0}$. If a vertex has more than one basic set that defines it then the polyheder is said to be degenerate.

The Faybusovich vector field is a projection of the gradient of the linear cost function onto the constraint set. Let $f(\mathbf{x})=\mathbf{c}^T\mathbf{x}$. We denote this gradient by grad f. Its explicit form is

$$\text{grad } f(\mathbf{x})=[D(\mathbf{x})-D(\mathbf{x})\,A^T(AD(\mathbf{x})\,A^T)^{-1}\,AD(\mathbf{x})]\,\mathbf{c}. \tag{6.3}$$

It is clearly an elementary formula as in the definition (4.1). A complete characterization of the dynamics of this vector field is as follows.

THEOREM 6.1 [38]. *Let* $(A, \mathbf{b}, \mathbf{c})$ *be the data for a non-degenerate linear program with a unique bounded solution. Then*:

1. *The faces of the polyheder are invariant sets of the dynamics induced by grad f.*

2. *The set of fixed points of grad f coincides with the vertices of the polyheder.*

3. *On an interior initial condition, the dynamics converges exponentially to the maximal vertex of the LP problem.*

We define $n-m$ vectors $\boldsymbol{\mu}_i \in \mathbb{R}^n$ which correspond to the $n-m$ coordinates of a set N such that $|N| = n-m$. Let $\mathbf{e}_n$ be the standard basis of $\mathbb{R}^n$. Define

$$\boldsymbol{\mu}_i = \mathbf{e}_i + \sum_{j \in B} \alpha_{ji} \mathbf{e}_j, \qquad i \in N, \tag{6.4}$$

where

$$\alpha_{ji} = -(A_B^{-1} A_N)_{ij}. \tag{6.5}$$

The vectors $\{\boldsymbol{\mu}_i\}_{i \in N}$ are perpendicular to the rows of A and are parallel to the faces of the polyheder defined by the constraints.

The ODE $\dot{\mathbf{x}} = \operatorname{grad} f$ has an analytical solution which describes the evolution of $n-m$ independent variables,

$$x_i(t) = x_i(0)\, e^{-\Delta_i t - \sum_{j \in B} \alpha_{ji} \ln(x_j / x_j(0))}, \qquad i \in N, \tag{6.6}$$

where $x_i(0)$ are the components of the initial condition and

$$\begin{aligned} \Delta_i &= -\langle \mathbf{c}, \boldsymbol{\mu}_i \rangle, \qquad i \in N, \\ &= -c_i + \sum_{j \in B} \alpha_{ji} c_j. \end{aligned} \tag{6.7}$$

Remark 6.1. If the basic set B in Eq. (6.6) is chosen to be a basic set corresponding to a maximum vertex, then all the Δ_i are positive. Thus it is evident that the analytical solution is only a formal one and does not provide an answer to the LP problem.

Remark 6.2. If $\mathbf{c}$ is perpendicular to a face of the polyheder then the maximum vertex of the LP problem is not unique and some of the Δ_i are 0. In such a case the flow converges to the optimal *face* of the polyheder, i.e., to a convex combination of the optimal vertices.

The computation time of the system is determined by the system size dependence of the constants Δ_i and α_{ij}. If, for example, the Δ_i are small then long computation times will be required. For general instances of LP these constants can be exponentially small, for a worst case exponential computation time. Exponentially small Δ_i are possible since Δ_i are defined by a matrix inverse which can yield exponentially small quantities [44]. In [45] we show that under a probability distribution on LP instances such "bad" instances happen with vanishing probability as n tends to infinity. Many problems in combinatorial optimization are expressible as linear programming problems which are often simpler to solve than general linear programming problems. Examples include maximum network flow (MAXFLOW) and bipartite matching [46, 47]. Thus, to make the Faybusovich vector field the basis of an algorithm which is efficient even in the worst case, we apply it to LP formulations of MAXFLOW and other related problems and show that these have polynomially convergent Faybusovich flows. Polynomial convergence occurs in these cases since the constants Δ_i and α_{ji} are small integers. Integrality of the Δ_i will be guaranteed when the constraint matrix is totally unimodular.

DEFINITION 6.1. An integer matrix A is called *totally unimodular* (TUM) if every square nonsingular submatrix B of A satisfies $\det B = \pm 1, 0$.

For a problem defined by a totally unimodular matrix, the solution polyheder is integral when $\mathbf{b}$ is an integer, as can be seen from Eq. (6.2). The next theorem gives conditions for TUM.

THEOREM 6.2 [46, 47]. *Let A be a matrix which is either*

1. *the node edge incidence matrix of a directed graph or*
2. *the node edge incidence of an undirected bipartite graph*;

then A is TUM. This includes the LP formulations of MAXFLOW, maximum weighted bipartite matching, and shortest path problems.

More general conditions for unimodularity are also known [47, 46]. Total unimodularity yields the following.

LEMMA 6.1. *Let A be a constraint matrix which is TUM, such that each column has at the most k nonzero entries, and let B and N be basic and non-basic sets respectively; then*

1. $|\alpha_{ji}| = |(A_B^{-1} A_N)_{ji}| \leqslant k$ *for every* $i \in B, j \in N$;
2. Δ_i *is integer for every* $i \in N$.

Proof. The first result is true since α_{ji} is a multiplication of two vectors whose components are $\{\pm 1, 0\}$, one of which has at most k nonzero

components. Δ_i are integer as a scalar product of two vectors with integer components. ▮

6.1. *Solving MAXFLOW*

DEFINITION 6.2. A network $\mathcal{N} = (s, t, V, E, \mathbf{b})$ is a directed graph (V, E) with a source $s \in V$ with no incoming edges, a terminal $t \in V$ with no outgoing edges, and an integer nonzero capacity b_i for edge i. Each edge is assigned a variable x_i, which is the flow through it. For convenience we add an additional edge with unlimited capacity from t to s. Let (V, E) be the resulting graph, denote $q = |V|$, $p = |E|$, and assign the variable x_p to the flow on the edge from t to s. An assignment of values $\mathbf{x} \in \mathbb{R}^p$ is a valid flow for a network $\mathcal{N}$ if

$$\mathbf{0} \leqslant \mathbf{x} \leqslant \mathbf{b}, \tag{6.8}$$

and it satisfies conservation of flow at each vertex $k \in V$,

$$\sum_{i \in \text{in}(k)} x_i - \sum_{i \in \text{out}(k)} x_i = 0, \tag{6.9}$$

where $\text{in}(k)/\text{out}(k)$ are the set incoming/outgoing edges, respectively, of vertex k. The flow through the network is given by the value of the variable x_p. The objective is to maximize x_p subject to the constraints (6.8) and (6.9).

We now express this problem as an LP problem in standard form. Let A' be the node-edge incidence matrix of the directed graph. The matrix A' has a row for each vertex and a column for each edge and is defined by $A'_{ij} = 1$ if edge j leaves node i or $A'_{ij} = -1$ if edge j enters node i, and is zero otherwise. The MAXFLOW problem is now expressed as the LP problem (not in standard form)

$$\begin{aligned} & & \max\ & x_p \\ \text{subject to:} \quad & A'\mathbf{x} & = & \mathbf{0} \\ & \hat{\mathbf{x}} & \leqslant & \mathbf{b} \\ & \mathbf{x} & \geqslant & \mathbf{0}, \end{aligned}$$

where $\hat{\mathbf{x}} = (x_1, ..., x_{p-1})$ (x_p has no upper bound). Therefore the vector $\mathbf{c}$ has a single nonzero component $c_p = 1$. The above formulation is made into an LP in standard form by adding the slack variables $x_{p+1}, ..., x_{2p-1}$,

with the constraints $x_i + x_{i+p} = b_i$, $x_{i+p} \geqslant 0$, $i = 1, ..., p-1$. The constraint matrix has now the form

$$A = \begin{pmatrix} A' & 0_{q \times p-1} \\ I_{p-1}\, 0_{p-1 \times 1} & I_{p-1} \end{pmatrix}, \tag{6.10}$$

where A' is the $q \times p$ node edge incidence matrix of (V, E), I_k is the $k \times k$ identity matrix, and $0_{k \times l}$ is a zero matrix of the appropriate dimensions. The right-hand side of the constraints is the vector $\mathbf{b} \in \mathbb{N}^{q+p-1}$,

$$\mathbf{b} = (0, ..., 0, b_1, ..., b_{p-1})^T. \tag{6.11}$$

For integer capacities the bit size of an instance of MAXFLOW is characterized by the number

$$L = \sum_{i=1}^{p-1} (\log b_i + 1). \tag{6.12}$$

The additional slack variable constraints do not affect the total unimodularity of the constraint matrix [46]. The matrix A is not full rank since the flow conservation equations represented by A' are not independent (adding the rows of A' gives 0). To make them so, it is sufficient to remove, say, the first row [47, 48]. We still denote the resulting matrix, which has $q + p - 2$ rows, by A. Anticipating the addition of another variable, we denote

$$n = 2p, \qquad m = p + q - 2, \tag{6.13}$$

and the problem is now in the standard form (6.1). We provide an interior initial condition by using the big-M method [49, 50]: we add to the variables $(x_1, ..., x_{n-1})$ an additional variable x_n with $c_n = -2^{L+1}$. A column A_n is added to A such that $(\mathbf{e}, 1) = (1, ..., 1)^T \in \mathbb{R}^n$ is a feasible point of the extended problem. The column A_n is required to satisfy

$$A\mathbf{e} + A_n = \mathbf{b}, \tag{6.14}$$

from which we have

$$A_n = \mathbf{b} - A\mathbf{e}. \tag{6.15}$$

The resulting constraint matrix $(A\ A_n)$ is not TUM. However, we can still guarantee integer Δ_i. First we note that it can be assumed that the solution

polyheder of the MAXFLOW problem is nonempty, because it can be made so by adding an edge from s to t with unit capacity. Thus the column corresponding to the variable x_n never enters into a basis B which corresponds to a maximum solution, therefore A_B is still TUM, and since A_N is still integer the Δ_i are integer.

The polyheder defined by the above constraints is in general highly degenerate [46, 47]. Therefore, in order to apply the Faybusovich vector field we perturb the right-hand side in order to lift the degeneracy

$$\tilde{\mathbf{b}} = \mathbf{b} + \delta\mathbf{b}, \tag{6.16}$$

where

$$\delta b_i = \frac{1}{n^2} 2^{-i}. \tag{6.17}$$

In the following we use the notation $\tilde{\mathbf{x}}$ to denote variables of the perturbed problem. Given a basic set B, the solution relative to B is perturbed by

$$\tilde{\mathbf{x}}_B = \mathbf{x}_B + \delta\mathbf{x}_B = \mathbf{x}_B + A_B^{-1}\,\delta\mathbf{b}. \tag{6.18}$$

With this perturbation we have

CLAIM 6.1. *For every choice of a basic set B such that $n \notin B$, the components of $\delta\mathbf{x}_B$ from* (6.18) *satisfy*

$$\frac{1}{n^2 2^m} \leqslant |\delta x_j| \leqslant \frac{1}{n^2}, \tag{6.19}$$

and in particular

$$\frac{1}{n^2 2^m} \leqslant \tilde{x}_j. \tag{6.20}$$

Proof. The assumption that $n \notin B$ takes into account that A_B does not include the column A_n. For such B the matrix A_B is TUM. The components of $\delta\mathbf{x}_B$ are of the form

$$\delta x = \mathbf{a}^T\,\delta\mathbf{b}, \tag{6.21}$$

where $\mathbf{a}^T \in \{-1, 0, 1\}^m$ is a row of A_B^{-1}. Using the form of $\delta\mathbf{b}$,

$$\delta x = \frac{1}{n^2} \sum_{i=1}^{m} \frac{a_i}{2^i}, \tag{6.22}$$

and it is immediate to verify that $|\delta x| \leqslant 1/n^2$. Without loss of generality, assume that $a_1 = 1$. Since $\sum_{i=2}^{m} 2^{-i} < 1/2$, δx will be smallest if all $a_i = -1$ for $i > 1$, and it is again immediate to verify that $|\delta x| \geqslant 1/n^2 2^m$. The validity of Eq. (6.20) follows from the integrality of the components of $\mathbf{x}_B$. ▮

We note that the matrix A could have been perturbed instead, but such a perturbation is harder to control.

6.2. *Complexity*

In this section we derive the complexity of solving the MAXFLOW problem by using the Faybusovich vector field. We recall from Remark 5.1 that for gradient flows a simple halting criterion can be used. Since we are considering problems with integer capacities, the maximum flow is integer, and if

$$f(\mathbf{x}^*) - f(\tilde{\mathbf{x}}) < 1/2 \tag{6.23}$$

where $\mathbf{x}^*$ is a maximizer of the unperturbed problem, then the computation can be halted. In the maxflow problem this translates to $x_p^* - \tilde{x}_p < 1/2$, and in view of the small effect of the perturbation it will be enough to obtain

$$\tilde{x}_p^* - \tilde{x}_p < 1/4, \tag{6.24}$$

where $\tilde{x}_p^*$ is the asymptotic value of $\tilde{x}_p$. From now on we drop the tilde notation, with the understanding that we are dealing with the perturbed problem. The complexity of the algorithm is a lower bound on the time required for the above to hold.

In general the flow will converge to a face of the polyheder and not to a vertex. A face is defined by a set N', $|N'| < n - m$, of indices such that $\mathbf{x}_{N'} = \mathbf{0}$. The next lemma shows that when $\mathbf{x}_{N'}$ are close to zero the computation can be halted. First we recall a few definitions and results regarding network flows. A *cut* in a network $\mathcal{N}$ is a partition of V into two sets V_1, V_2 such that $s \in V_1$ and $t \in V_2$. We denote by $V_i \to V_j$ for $i \neq j$ the set of edges from V_i to V_j. The *capacity* of a cut is defined as $\sum_{i \in V_1 \to V_2} b_i$. It is well known that the maximum flow equals the capacity of the cut with

minimum capacity, and that given such a minimum cut V_1, V_2 the flow into t equals the flow across the cut,

$$x_p = \sum_{i \in V_1 \to V_2} x_i - \sum_{i \in V_2 \to V_1} x_i. \tag{6.25}$$

We now state

LEMMA 6.2. *Let N' be the maximal set such that $\mathbf{x}_{N'}$ converge to zero; then if $\mathbf{x}_{N'} = O(1/n^2)$ then $x_p^* - x_p \leqslant 1/4$.*

Proof. Let V_1, V_2 be a minimum capacity cut in $\mathcal{N}$. In any maximum flow, the edges from V_1 to V_2 are saturated (i.e., $x_i = b_i$), the corresponding slack variables x_{i+p} are zero, and in edges from V_2 to V_1 the flow is zero. Using the relation $x_i + x_{i+p} = b_i$ we rewrite (6.25) as

$$x_p = \sum_{i \in V_1 \to V_2} (b_i - x_{i+p}) - \sum_{i \in V_2 \to V_1} x_i, \tag{6.26}$$

and since $x_p^* = \sum_{i \in V_1 \to V_2} b_i$ we have

$$x_p - x_p^* = -\sum_{i \in V_1 \to V_2} x_{i+p} - \sum_{i \in V_2 \to V_1} x_i. \tag{6.27}$$

All the variables in this equation belong to N', i.e., they eventually converge to 0, and thus we have

$$x_p^* - x_p \leqslant 2p \max \mathbf{x}_{N'}. \tag{6.28}$$

Thus $x_p^* - x_p \leqslant 1/4$ if $\mathbf{x}_{N'} = O(1/n^2)$. ∎

We are now ready to prove

THEOREM 6.3. *The CTC for MAXFLOW has complexity $T = 7(m^2 + L)$.*

Proof. The complexity of the algorithm is the time required to obtain $\mathbf{x}_{N'} \leqslant 1/n^2$. Using the analytical solution equation (6.6),

$$1/n^2 \geqslant \exp\left(-t + \sum_{j \in B} \alpha_{ji} \ln x_j \right), \tag{6.29}$$

where B is a basic set of the optimal solution. Here we have used the fact that the Δ_i are positive integers. To compute the bound on t we first find a bound on

$$\beta_i = \left| \sum_{j \in B} \alpha_{ji} \ln x_j \right|. \tag{6.30}$$

Using $|\alpha_{ji}| \leqslant 3$ we have

$$\beta_i \leqslant 3 \sum_{j \in B} |\ln x_j|. \tag{6.31}$$

We partition the set B into two sets, a set B_1 of indices that correspond to coordinates that converge to "small" values (less than 1) and a set B_2 of indices that correspond to coordinates that converge to values larger than 1. For $i \in B_1$ we have from (6.20) that $|\ln x_i| \leqslant 2m$. For $i \in B_2$, $i \neq p$, we have $1 \leqslant x_i \leqslant b_i$; i.e., $\sum_{i \in B_2,\, i \neq p} |\ln x_j| \leqslant \sum_j \ln b_j \leqslant L$. The variable x_p is not bounded; however, by flow conservation it is less than the sum of the other variables, i.e., $\ln x_p \leqslant L$. By combining these bounds we obtain

$$\beta_i \leqslant 6m^2 + 6L. \tag{6.32}$$

We now have

$$1/n^2 \geqslant \exp(-t + 6(m^2 + L)) \tag{6.33}$$

which is satisfied for $t \geqslant 7(m^2 + L)$. ∎

Remark 6.3. The above analysis was carried out specifically for the LP formulation of the MAXFLOW problem. Similar complexity bounds will hold for other problems defined by TUM constraint matrices.

Remark 6.4. Parallel algorithms for the MAXFLOW problem are discussed for example in [51]. The best algorithms described there have time complexity $O(|V|^2 \log |V|)$ and use $O(\sqrt{|E|})$ processors. In our notation this is $O(n \log n)$, which is better than our algorithm.

7. COMPLEXITY CLASSES

We have defined time complexity for a class of continuous time systems.[5] In the following we compare complexity in our model with the classical theory, so we consider integer or rational inputs. The operations which define a formula make it computable in NC. However, for the definition of the logarithmic time complexity class a stricter form of uniformity is required, e.g., allowing only those operations which are computable in AC_0 [8].

DEFINITION 7.1. A problem is said to be in CLOG (continuous log) if it has a CTC with a polynomial number of variables and logarithmic time complexity.

[5] There seems to be no obvious counterpart for the space resource in these systems. One can consider the variables as memory units, but they also play the part of processing units.

This is a counterpart of the classical NC_1 [8]. We have shown that MAX (for integer inputs) is in CLOG. The counterpart of the classical P is CP:

DEFINITION 7.2. A problem is said to be in CP (continuous P) if it has a CTC with a polynomial number of variables and polynomial time complexity.

In Section 6 we have shown a CTC for the maximum network flow problem which places it in CP. The maximum network flow problem is P-complete relative to logspace Turing reductions [52]. Relying on this result we argue that $P \subseteq CP$: If we use the Turing reductions from a P problem to maximum network flow then all efficient Turing computations can be performed polynomially in our framework if we allow the pre-processing required for the reduction. Using Turing reductions which are outside our model might be considered unsatisfactory, but at the moment we know how to solve only specific problems in our framework if we do not use a pre-processing stage. In this context we mention the related result that approximating the fixed point of an NC contracting mapping is P-complete [37] and note that contracting maps converge exponentially to fixed points. This means that an NC vector field is sufficient to obtain the classical P. The dynamical system used to solve the maximum network flow problem is a gradient flow, so the subclass of CP which consists of analytically solvable gradient flows can be considered as computationally powerful as P.

As of yet we have no argument for the inclusion $CP \subseteq P$. The analytical solution is no help in computing the attractor of the system, as exemplified with the MAX and MAXFLOW problems. However, we believe that a polynomial time simulation of the ODE with some numerical integration scheme should be possible for the class of vector fields considered in this paper.

8. DISCUSSION

In the following we suggest possible generalizations and variations of the model presented here. With no knowledge on bounds on the time to reach the trapping region we can resort to probabilistic verification of an attractor: when it is suspected that a fixed point is approached, a number of trajectories are initiated in an ε-ball around the trajectory. If this ball shrinks then with high probability the fixed point is an attractor. If the ball has expanded in some direction, then the fixed point is a saddle point. This yields a Co-RP type of complexity class which is applicable to gradient flows, for example [39]. Here we still need to know some initial condition

which leads to the desired attractor. When even this information is not known, one can nondeterministically choose an initial condition in the appropriate basin of attraction.

In this paper only fixed point attractors were considered. In another paper [39] computation of chaotic attractors is discussed. Such attractors are found to be computable efficiently by means of nondeterminism (in a different sense than the one mentioned above). The inherent difference between fixed points and chaotic attractors has lead to the conjecture that for the dynamical systems complexity classes $CP \neq CNP$ [39]. This may shed light on the P vs NP question in standard complexity theory.

In this paper we defined the input to be the parameters of the CTC, while the vector field with an initial condition constitutes the continuous algorithm. Another input convention can be considered, namely treating the initial condition as input. Such an approach is motivated by classification problems and attractor models of associative memory [1]. This is also how a Turing machine works: the inputs (with the initial state) are the initial configuration for a Turing machine, and the computation converges to a fixed point which corresponds to the decision accept/reject. Treating the initial condition as input in our framework has the problem that the computation time for inputs near the basin boundary is arbitrarily long. Boundaries present problems of decidability in other models as well: for example, in Ko's framework [53] an algorithm is allowed to err for a set of small measure in the vicinity of the boundary. In this framework there exist boundary sets (Julia sets) of computable functions that are not recursively approximable [54]. Even in a model of computation over the real numbers, such as the BSS model [11], there are boundary sets that are not recursive. Extensions of our continuous time computer and related models are currently under consideration.

ACKNOWLEDGMENTS

The authors thank E. Sontag, K. Ko, C. Moore, J. Crutchfield, T. Artstein, and an anonymous referee for their comments on this work. This work was supported in part by the U.S.–Israel Binational Science Foundation (BSF), by the Israeli Ministry of Arts and Sciences, by the Fund for Promotion of Research at the Technion, and by the Minerva Center for Nonlinear Physics of Complex Systems.

REFERENCES

1. J. Hertz, A. Krogh, and R. Palmer, "Introduction to the Theory of Neural Computation," Addison–Wesley, Redwood City, CA, 1991.
2. A. Cichocki and R. Unbehauen, "Neural Networks for Optimization and Signal Processing," Wiley, New York, 1993.

3. X. B. Liang and J. Wang, A recurrent neural network for nonlinear optimization with a continuously differentiable objective function and bound constrants, *IEEE Trans. Neural Networks* (2000).
4. C. Mead, "Analog VLSI and Neural Systems," Addison–Wesley, Reading, MA, 1989.
5. C. Koch and H. Li, "Vision Chips: Implementation of Vision Algorithms with Analog VLSI Circuits," IEEE Computer Society, Piscataway, NJ, 1994.
6. J. Indiveri, Winner-take-all networks with lateral excitation, *Analog Integrated Circuits Signal Process.* **13** (1997), 185–193.
7. U. Helmke and J. B. Moore, "Optimization and Dynamical Systems," Springer-Verlag, London, 1994.
8. C. Papadimitriou, "Computational Complexity," Addison–Wesley, Reading, MA, 1995.
9. E. Ott, "Chaos in Dynamical Systems," Cambridge Univ. Press, Cambridge, MA, 1993.
10. L. Blum, M. Shub, and S. Smale, On a theory of computation and complexity over the real numbers: NP completeness, recursive functions, and universal machines, *Bull. Amer. Math. Soc.* **21** (1989), 1–46.
11. L. Blum, F. Cucker, M. Shub, and S. Smale, "Complexity and Real Computation," Springer-Verlag, New York/Berlin, 1999.
12. J. von Neumann, "Collected Works" (A. H. Taub, Ed.), Vol. A, Pergamon, Oxford, 1961.
13. C. Moore, Unpredictability and undecidability in dynamical systems, *Phys. Rev. Lett.* **64** (1990), 2354–2357.
14. P. Koiran, M. Cosnard, and M. Garzon, Computability with low-dimensional dynamical systems, *Theoret. Comput. Sci.* **132** (1994), 113–128.
15. P. Koiran and C. Moore, Closed-form analytic maps in one and two dimensions can simulate universal Turing machines, *Theoret. Comput. Sci.* **210** (1999), 217–223.
16. R. W. Brockett, Hybrid models for motion control systems, *in* "Essays in Control: Perspectives in the Theory and Its Applications" (H. L. Trentelman and J. C. Willems, Eds.), pp. 29–53, Birkhäuser, Boston, 1993.
17. M. S. Branicky, Analog computation with continuous ODEs, *in* "Proceedings, IEEE Workshop on Physics and Computation," pp. 265–274, IEEE, Dallas, TX, 1994.
18. E. Asarin, O. Maler, and A. Pnueli, Reachability analysis of dynamical systems with piecewise-constant derivatives, *Theoret. Comput. Sci.* **138** (1995), 35–66.
19. P. Orponen, A survey of continuous time computation theory, *in* "Advances in Algorithms, Languages and Complexity," Kluwer Academic, Dordrecht, 1997.
20. C. E. Shannon, Mathematical theory of the differential analyzer, *J. Math. Phys. Mass. Institute Technol.* **20** (1941), 337–354.
21. M. B. Pour-El, Abstract computability and its relation to the general purpose analog computer (some connections between logic, differential equations and analog computers), *Trans. Amer. Math. Soc.* **199** (1974), 1–29.
22. M. L. Campagnolo, C. Moore, and J. F. Costa, Iteration, inequalities and differentiability in analog computers, *Complexity*, to appear.
23. O. Bournez, Achilles and the Tortoise climbing up the hyper-arithmetical hierarchy, *Theoret. Comput. Sci.* **210**, No. 1 (6 Jan. 1999), 21–71.
24. C. Moore, Recursion theory on the reals and continuous-time computation, *Theoret. Comput. Sci.* **162** (1996), 23–44.
25. J. J. Hopfield and D. W. Tank, Neural computation of decisions in optimization problems, *Biological Cybernetics* **52** (1985), 141–152.

26. J. Sima and P. Orponen, A continuous-time hopfield net simulation of discrete neural networks, Tech. Rep. 773, Academy of Sciences of the Czech Republic, 1999.
27. R. W. Brockett, Dynamical systems that sort lists, diagonalize matrices and solve linear programming problems, *Linear Algebra Appl.* **146** (1991), 79–91.
28. L. Faybusovich, Hamiltonian structure of dynamical systems which solve linear programming problems, *Physica D* **53** (1991), 217–232.
29. R. Feynman, Quantum mechanical computers, *Found. Phys.* **16**, No. 6 (1986), 507–531 [appeared originally in *Optics News* (Feb. 1985)].
30. H. T. Siegelmann and E. D. Sontag, Analog computation via neural networks, *Theoret. Comput. Sci.* **131** (1994), 331–360.
31. H. T. Siegelmann, Computation beyond the Turing limit, *Science* **268**, No. 5210 (Apr. 28, 1995), 545–548.
32. H. T. Siegelmann, A. Ben-Hur, and S. Fishman, Computational complexity for continuous time dynamics, *Phys. Rev. Lett.* **83**, No. 7 (1999), 1463–1466.
33. M. Hirsch and S. Smale, "Differential Equations, Dynamical Systems and Linear Algebra," Academic Press, New York, 1974.
34. J. Guckenheimer and P. Holmes, "Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields," Springer-Verlag, New York, 1983.
35. M. Zak, Terminal attractors in neural networks, *Neural Networks* **2** (1989), 259–274.
36. B. R. Hunt, T. Sauer, and J. A. Yorke, Prevalence: A translational-invariant "almost every" on infinite dimensional spaces, *Bull. Amer. Math. Soc.* **27**, No. 2 (1992), 217–238.
37. H. J. Hoover, Real functions, contraction mappings and P-completeness, *Inform. and Comput.* **93**, No. 2 (1991), 333–349.
38. L. Faybusovich, Dynamical systems which solve optimization problems with linear constraints, *IMA J. Math. Control Inform.* **8** (1991), 135–149.
39. H. T. Siegelmann and S. Fishman, Computation by dynamical systems, *Physica D* **120** (1998), 214–235.
40. Y. Shiloach and U. Vishkin, Finding the maximum, merging and sorting in a parallel computation model, *J. Algorithms* **2** (1981), 88–102.
41. S. Smale, Some remarks on the foundations of numerical analysis, *SIAM Rev.* **32** (1990), 211–220.
42. J. P. Dedieu, Approximate solutions of numerical problems, condition number analysis and condition number theorem, *in* "The Mathematics of Numerical Analysis, Utah, July 1995" (J. Renegar, M. Shub, and S. Smale, Eds.), pp. 263–283, American Mathematical Society, Providence, RI, 1995.
43. J. Renegar, Incorporating condition measures into the complexity theory of linear programming, *SIAM J. Optim.* **5**, No. 3 (1995), 506–524.
44. F. R. Gantmacher, "The Theory of Matrices," Chelsea Publishing, New York, 1960.
45. A. Ben-Hur, J. Feinberg, S. Fishman, and H. T. Siegelmann, The complexity of solving linear programming with a differential equation, manuscript.
46. C. H. Papadimitriou and K. Steiglitz, "Combinatorial Optimization," Prentice–Hall, Englewood Cliffs, NJ, 1982.
47. G. L. Nemhauser and L. A. Wolsey, "Integer and Combinatorial Optimization," Wiley, New York, 1988.
48. M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali, "Linear Programming and Network Flows," Wiley, New York, 1990.

49. S. J. Wright, "Primal-Dual Interior-Point Methods," SIAM, Philadelphia, PA, 1997.
50. R. Saigal, "Linear Programming," Kluwer Academic, Boston, 1995.
51. A. V. Goldberg, Parallel algorithms for network flow problems, *in* "Synthesis of Parallel Algorithms" (J. H. Reif, Ed.), Morgan Kaufmann, San Mateo, CA, 1993.
52. L. M. Goldschlager, R. A. Shaw, and J. Staples, The maximum flow problem is log space complete for P, *Theoret. Comput. Sci.* **21** (1982), 105–111.
53. A. Chou and K. Ko, Computational complexity of two-dimensional regions, *SIAM J. Comput.* **24** (1995), 923–947.
54. K. Ko, On the computability of fractal dimensions and Housdorff measure, *Ann. Pure Appl. Logic* **93** (1998).